Embedded Vehicle Detection with YOLOv5n on ESP32-S3 for Urban Traffic Applications

Danilo J. F. de Souza[1], **Carlos M. S. Figueiredo**[1]

[1]LSI–Lab. de Sistemas Inteligentes
Universidade do Estado do Amazonas
Av. Darcy Vargas, 1200 – Manaus – Amazonas

{djfds.eng22, cfigueiredo}@uea.edu.br

Abstract. *This paper presents the implementation and evaluation of YOLOv5n models for vehicle detection on urban roads using an ESP32-S3 embedded device. The models were trained at different input resolutions, converted to C code using SynapEdge, and evaluated in terms of offline detection performance and embedded inference time. The model trained and evaluated at 224×224 pixels achieved the best detection performance, with an mAP50 of 0.683, but required the highest inference time on the ESP32-S3. In the evaluated floating-point implementation, the 160×160 configuration represented an intermediate operating point between detection performance and computational cost, with an mAP50 of 0.468 and an average inference time of 13.126 s. The results show that lightweight object detection models can be executed on low-cost embedded devices, although input resolution strongly affects both accuracy and processing time.*

1. Introdução

O congestionamento do tráfego urbano é um problema recorrente em diversas cidades pelo mundo, afetando a mobilidade, a segurança e a qualidade de vida da população. Segundo estudo realizado pelo Instituto Cordial em parceria com a Uber, acidentes de trânsito são mais frequentes quando há congestionamento: a chance de ocorrer um acidente é 3,7 vezes maior em congestionamentos leves e 6,3 vezes maior em congestionamentos pesados [Uber Newsroom 2020]. Dessa forma, evidencia-se que a gestão do tráfego de maneira eficaz se tornou um fator vital para o bem-estar social, especialmente nas capitais, regiões metropolitanas e centros urbanos com elevada densidade de veículos e pessoas.

Entre os métodos utilizados para detecção de objetos, destacam-se os modelos da família YOLO, amplamente empregados em tarefas de localização e classificação de objetos em imagens [Sapkota and Karkee 2025]. Contudo, embora esses modelos sejam voltados à detecção em tempo real em plataformas computacionais convencionais, sua execução em microcontroladores permanece desafiadora devido às restrições de memória, processamento e armazenamento. Assim, aplicações embarcadas de baixo custo podem ser mais adequadas a cenários de baixa frequência de aquisição, contagem periódica ou amostragem esparsa, especialmente quando não são aplicadas otimizações adicionais como quantização.

Dessa forma, este trabalho avalia a execução embarcada de modelos YOLOv5n para detecção de veículos em um microcontrolador ESP32-S3. Para isso, modelos são treinados em diferentes resoluções de entrada, convertidos para código C com o SynapEdge e avaliados quanto à precisão, revocação, mAP50 e tempo de inferência no dispositivo. O objetivo é identificar, na implementação em ponto flutuante avaliada, a resolução que oferece o melhor compromisso observado entre qualidade de detecção e custo computacional.

2. Fundamentos e Trabalhos Relacionados

Em meio ao crescente avanço tecnológico ao longo dos anos, foram apresentados novos conceitos e pesquisas relevantes para o desenvolvimento de soluções voltadas ao monitoramento e à gestão de ambientes urbanos. Entre essas abordagens, destaca-se o conceito de "Cidades Inteligentes", definido como um modelo de desenvolvimento urbano que visa a satisfação dos seus habitantes, empresas, visitantes, organizações e administradores através de serviços tecnológicos eficientes que contribuem para uma melhor qualidade de vida e sustentabilidade ambiental [Leite and Awad 2012]. Na área da computação, essa ideia está intrinsecamente relacionada às áreas de Aprendizado de Máquina e Visão Computacional, que permitem a extração de informações relevantes a partir de grandes volumes de dados.

Tratando-se de um campo da inteligência artificial, o Aprendizado de Máquina permite que sistemas computacionais aprimorem seu desempenho através da experiência, sem que sejam programados explicitamente para atividades específicas [Brink et al. 2016]. A Visão Computacional, por sua vez, permite que computadores interpretem informações visuais presentes em imagens digitais, vídeos e outras entradas visuais. Por meio de técnicas baseadas em redes neurais e aprendizado profundo, é possível identificar, classificar e localizar objetos em cenas complexas, tornando essa área especialmente relevante para aplicações de monitoramento urbano [Prince 2012].

Explorando essas capacidades, diversos estudos recentes têm aplicado tais tecnologias ao monitoramento de tráfego urbano. [Loureiro et al. 2025] utilizaram YOLOv11 Medium em imagens de câmeras públicas de rodovias brasileiras, obtendo mAP@0,5:0,95 de 0,7372 e taxa de 94 FPS para contagem e visualização de veículos em dashboard. [Ashkanani et al. 2025] empregaram YOLOv11 para detecção de veículos e ajuste dinâmico dos tempos de verde em cruzamentos semaforizados, obtendo mAP50 de 0,951 com execução em Raspberry Pi. Em abordagem complementar, [Zhao et al. 2020] propuseram o modelo T-GCN, que combina redes convolucionais em grafo com unidades recorrentes com porta para capturar dependências espaciais e temporais na previsão de velocidade em vias urbanas.

No contexto de dispositivos embarcados, [Moosmann et al. 2024] avaliaram variantes quantizadas de uma rede da família YOLO em cinco plataformas de ultrabaixa potência, caracterizando o impacto da resolução de entrada, do número de classes e da largura das camadas sobre latência, energia e qualidade de detecção. Em hardware da família ESP32, [Chang et al. 2025] investigaram detecção embarcada com TensorFlow Lite Micro, analisando o efeito do tamanho da imagem sobre o tempo de classificação. Esses trabalhos evidenciam os desafios de executar modelos de detecção em dispositivos de borda de baixo custo.

Diferentemente desses estudos, este trabalho analisa o impacto da resolução de entrada na qualidade de detecção e no tempo de inferência de modelos YOLOv5n convertidos para C e executados em um ESP32-S3, visando identificar uma configuração adequada para detecção embarcada de veículos em cenários de baixa cadência.

3. Materiais e Métodos

A metodologia adotada neste trabalho foi estruturada em etapas sequenciais, conforme ilustra a Figura 1. A princípio, foram realizadas pesquisas acerca da plataforma embarcada mais adequada para os objetivos do projeto, visando obter a melhor relação custo-benefício computacional (Seção 3.1). Então, definiu-se a ferramenta responsável pela conversão do modelo para linguagem C, *SynapEdge* (Seção 3.2). Posteriormente, foram realizados os ajustes necessários para viabilizar a execução da inferência na placa ESP32-S3 (Subseções 3.2.1 e 3.2.2). Na sequência, selecionou-se um dataset específico para tráfego urbano (Seção 3.3) para realizar o treinamento de modelos YOLOv5 nano em diferentes resoluções e configurações (Seções 3.4 e 3.5). Por fim, concluiu-se com a conversão dos modelos treinados para código C (Seção 3.6) e escolha das métricas obtidas em ambiente offline e embarcado (Seção 3.7).

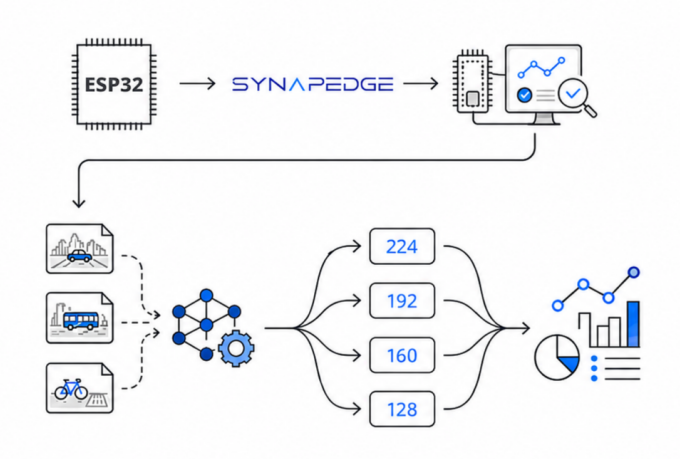


Figura 1. Fluxo de trabalho geral

3.1. Placa embarcada escolhida

A plataforma definida para os experimentos foi o microcontrolador ESP32-S3 N16R8, escolhido por seu baixo custo, disponibilidade de PSRAM e recursos computacionais mínimos suficientes para armazenar e executar um modelo de detecção de objetos. A placa possui frequência de operação de até 240 MHz, 16 MB de memória Flash, 8 MB de PSRAM e 512 KB de SRAM interna.

3.2. SynapEdge: conversor de modelos YOLOv5 para C

Para executar o modelo no microcontrolador, utilizou-se o SynapEdge [Shafi 2025], ferramenta de código aberto que converte modelos YOLOv5n para uma implementação em linguagem C compatível com plataformas ESP32-S3. O fluxo empregado consiste em treinar o modelo em PyTorch, exportá-lo para ONNX e convertê-lo para código C, posteriormente integrado ao projeto embarcado na Arduino IDE, conforme a Figura 2.

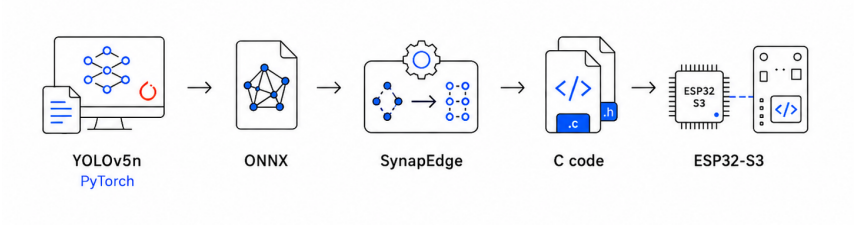


Figura 2. Fluxo de conversão e integração do modelo YOLOv5n.

3.2.1. Configuração e ajustes no código para a placa ESP32-S3

Após a geração do código, foram necessárias adaptações para permitir sua compilação e execução no ambiente *Arduino IDE*. O arquivo principal gerado em C foi renomeado para extensão `.cpp`, de modo a manter compatibilidade com o processo de compilação da IDE. Além disso, considerando a limitação da SRAM interna do ESP32-S3, os tensores intermediários da rede foram alocados dinamicamente na PSRAM, enquanto suas declarações estáticas foram comentadas no arquivo de cabeçalho. Esse ajuste foi necessário para reduzir o uso da memória interna e permitir a execução do modelo alvo no microcontrolador.

Ademais, o ambiente *Arduino IDE* também foi configurado para utilizar 16 MB de memória Flash, PSRAM habilitada e uma tabela de partição customizada, permitindo reservar maior espaço para a aplicação embarcada do que aquele disponível nas configurações padrão de partição.

3.2.2. Pré-processamento, inferência e pós-processamento

As imagens de entrada foram armazenadas em arrays C no arquivo `image.h`. Antes da inferência, realizou-se a conversão RGB565 para RGB888, redimensionamento por vizinho mais próximo, normalização para [0,1] e reorganização no formato `[1][3][H][W]`. Após a execução de *forward_pass()*, as predições foram filtradas por confiança de 0,30 e NMS por classe de 0,40, sendo enviadas pela interface serial com classe, confiança e coordenadas da caixa delimitadora.

3.3. Dataset utilizado e preparação dos dados

Neste trabalho, utilizou-se uma base de dados previamente construída por [Loureiro et al. 2025], composta por 1000 imagens JPEG de vias urbanas brasileiras localizadas em Manaus, Barueri, Chapada e Porto Velho, obtidas a partir de câmeras públicas de trânsito. A base possui 6932 anotações, separadas em quatro classes: carro, moto, ônibus e veículo, sendo esta última uma classe genérica que contempla caminhões, vans, ambulâncias, dentre outros.

Conforme observado por [Loureiro et al. 2025] e ilustrado na Figura 3, a distribuição das anotações entre as classes apresenta desbalanceamento, fator que deve ser considerado na interpretação das métricas de desempenho dos modelos. A seleção dessa fonte de dados justifica-se por apresentar maior proximidade com o domínio de

aplicação deste estudo, contendo cenas reais do tráfego urbano brasileiro e classes de veículos compatíveis com o objetivo de detecção embarcada.

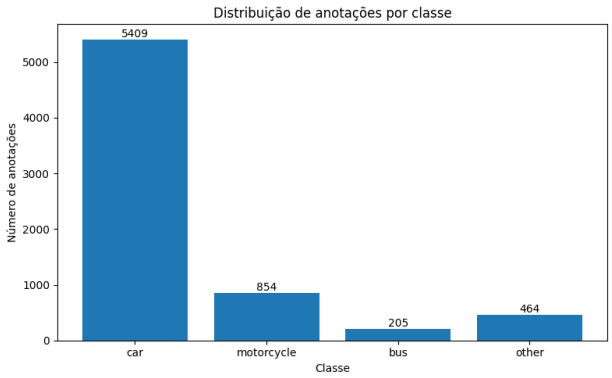


Figura 3. Histograma das anotações de classes.

Originalmente, o dataset encontrava-se organizado no formato YOLOv11. Para possibilitar sua utilização no treinamento com a arquitetura YOLOv5n, foi realizada a conversão das anotações para a notação esperada pelo YOLOv5. Além disso, a divisão original da base foi mantida, composta por 60% das imagens para treinamento, 10% para validação e 30% para teste, sendo preservada para manter a organização experimental definida na construção original do dataset.

No arquivo de configuração utilizado pelo YOLOv5, as classes foram representadas em inglês, seguindo a ordem: `car`, `motorcycle`, `bus` e `other`, em que `other` corresponde à categoria genérica originalmente denominada "veículo". Finalmente, o ajuste de resolução das imagens foi realizado diretamente durante o treinamento e a avaliação dos modelos, por meio do parâmetro `--img` do YOLOv5.

3.4. Configurações de treinamento e parametrização dos modelos YOLOv5n

Com o objetivo de avaliar o impacto da resolução de entrada na qualidade da detecção e na viabilidade de execução embarcada, treinou-se quatro modelos baseados na arquitetura YOLOv5n. As resoluções consideradas foram 128×128, 160×160, 192×192 e 224×224 pixels. A escolha dessas resoluções foi motivada pela necessidade de redução do custo computacional, considerando-se as restrições de memória e processamento do microcontrolador ESP32-S3.

Além dos modelos treinados diretamente nas resoluções alvo, também foi treinado um modelo YOLOv5n na resolução de 640×640 pixels, correspondente a uma configuração usual de treinamento da arquitetura YOLO. Esse modelo foi utilizado como referência adicional para investigar o efeito da redução da resolução apenas na etapa de avaliação, sem que o modelo tivesse sido treinado especificamente para as resoluções menores.

Em todos os treinamentos, utilizou-se a arquitetura YOLOv5n inicializada com pesos pré-treinados no conjunto COCO. Os modelos foram treinados por 200 épocas, com tamanho de lote igual a 64, utilizando o arquivo `data.yaml` do dataset personalizado para indicar os caminhos dos subconjuntos e a definição das classes. Adicionalmente, o parâmetro `--cache` foi empregado para otimizar o carregamento dos dados no ambiente

de treinamento. Os demais hiperparâmetros foram mantidos conforme os valores padrão do repositório YOLOv5 utilizado, visando manter o protocolo de treinamento constante entre as resoluções avaliadas e isolando principalmente o efeito da resolução de entrada.

3.5. Ambiente de treinamento

O treinamento dos modelos foi realizado em ambiente Google Colab, utilizando a GPU L4 e alta disponibilidade de memória RAM. Inicialmente, o repositório do YOLOv5 foi clonado no ambiente de execução e suas dependências instaladas. Em seguida, o conjunto de dados convertido para o formato YOLOv5 foi importado a partir do Google Drive, preservando a organização previamente definida em subconjuntos de treinamento, validação e teste.

Ao final de cada treinamento, os pesos gerados foram armazenados no Google Drive, permitindo sua posterior utilização nas etapas de validação, conversão e implementação embarcada. Para cada resolução avaliada, foi considerado o arquivo `best.pt`, correspondente ao melhor modelo obtido durante o processo de treinamento do YOLOv5n, conforme o critério de seleção implementado pelo framework. Destaca-se que o número reduzido de imagens de validação pode tornar a seleção do arquivo `best.pt` mais sensível a variações específicas desse subconjunto, especialmente por conta do desbalanceamento entre classes.

3.6. Conversão dos modelos para código C

Após o treinamento, cada modelo foi exportado do formato PyTorch para o formato ONNX, mantendo a mesma resolução de entrada utilizada durante o treinamento. Essa etapa teve como objetivo gerar uma representação intermediária da rede neural, permitindo sua conversão para uma implementação em linguagem C.

Em seguida, os arquivos ONNX foram convertidos através do projeto *SynapEdge* [Shafi 2025]. Como resultado, foram gerados arquivos em linguagem C contendo a estrutura da rede e os pesos do modelo, posteriormente incorporados ao projeto embarcado desenvolvido na *Arduino IDE*, possibilitando a execução da inferência diretamente na placa ESP32-S3.

3.7. Métricas e protocolo de avaliação

A avaliação dos modelos foi separada em duas etapas: avaliação offline e avaliação embarcada. A avaliação offline teve como objetivo mensurar a qualidade da detecção dos modelos treinados, enquanto a avaliação embarcada teve como objetivo verificar a viabilidade de execução dos modelos convertidos no microcontrolador ESP32-S3.

3.7.1. Avaliação offline

Este formato de análise foi dividido em dois cenários. No primeiro, cada modelo treinado nas resoluções de 128×128, 160×160, 192×192 e 224×224 pixels foi avaliado na mesma resolução utilizada durante seu treinamento. No segundo, um modelo treinado em 640×640 pixels foi avaliado nas mesmas resoluções alvo, permitindo comparar o desempenho de modelos treinados especificamente para resoluções reduzidas com o desempenho de um modelo treinado em resolução padrão e avaliado em imagens redimensionadas. Em ambos os casos, as seguintes métricas foram consideradas:

- **Precisão:** proporção de detecções corretas entre todas as detecções realizadas pelo modelo.
- **Revocação:** proporção de objetos corretamente detectados em relação ao total de objetos anotados.
- **mAP50:** média da precisão média das classes, considerando como correta uma detecção cuja interseção sobre união (IoU) em relação à anotação de referência seja igual ou superior a 0,50 [Everingham et al. 2010].

Destaca-se que as métricas offline foram obtidas por meio do script de validação do YOLOv5 aplicado ao subconjunto de teste, seguindo o protocolo padrão da ferramenta para cálculo de precisão, revocação e mAP50. Esse procedimento difere do pós-processamento embarcado, no qual foram definidos limiares fixos de confiança igual a 0,30 e NMS igual a 0,40 para filtragem das detecções enviadas pela interface serial.

3.7.2. Avaliação embarcada

A avaliação embarcada foi realizada após a conversão dos modelos para linguagem C e integração ao projeto desenvolvido para o ESP32-S3. Nessa etapa, a principal métrica considerada foi o **tempo de inferência**, medido com `micros()` no firmware, diretamente no ESP32-S3, considerando 10 execuções por resolução sobre imagens previamente carregadas na memória. Essa medição foi realizada sobre a etapa de execução da rede neural, após o carregamento da imagem na memória, não incluindo o tempo de aquisição da imagem por câmera. Finalmente, não foi adotada etapa específica de aquecimento nem descarte da primeira execução.

Dessa forma, a comparação entre os modelos foi conduzida considerando-se as métricas de qualidade de detecção obtidas na avaliação offline e o tempo de inferência constatado na avaliação embarcada. Essa análise permitiu investigar o compromisso entre desempenho preditivo e viabilidade computacional para cada uma das resoluções de 128×128, 160×160, 192×192 e 224×224 pixels.

4. Resultados e Discussão

Após execução dos treinamentos e avaliações dos modelos em diferentes resoluções, os resultados foram organizados de duas formas: avaliação offline, realizada no subconjunto de teste do dataset, e avaliação embarcada, realizada diretamente na placa ESP32-S3.

4.1. Desempenho offline dos modelos

A avaliação offline foi realizada com o objetivo de comparar a qualidade de detecção dos modelos YOLOv5n treinados em diferentes resoluções de entrada. A Tabela 1 apresenta os resultados dos modelos treinados e avaliados na mesma resolução. A Tabela 2 exibe os valores das métricas obtidas para o modelo treinado em resolução 640×640 e avaliado nas resoluções alvo definidas no estudo.

Tabela 1. Modelos treinados e avaliados na mesma resolução.

Resolução de treino	Resolução de avaliação	Precisão	Revocação	mAP50
224×224	224×224	0.736	0.641	0.683
192×192	192×192	0.713	0.593	0.637
160×160	160×160	0.503	0.522	0.468
128×128	128×128	0.462	0.415	0.390

Tabela 2. Modelo treinado em 640×640 e avaliado nas resoluções alvo.

Resolução de treino	Resolução de avaliação	Precisão	Revocação	mAP50
640×640	224×224	0.663	0.517	0.578
640×640	192×192	0.542	0.456	0.444
640×640	160×160	0.458	0.309	0.300
640×640	128×128	0.335	0.203	0.183

Observa-se que o modelo treinado e avaliado em 224×224 pixels obteve o melhor desempenho geral, com mAP50 de 0,683. A redução da resolução levou à queda progressiva das métricas, indicando perda de detalhes visuais relevantes para classificação e localização dos veículos. Além disso, os modelos treinados diretamente nas resoluções alvo superaram o modelo treinado em 640×640 e avaliado em resoluções reduzidas. Assim, nos experimentos realizados, treinar diretamente na resolução efetiva de operação mostrou-se mais adequado para inferência em baixa resolução. Contudo, como o modelo de 640×640 também foi treinado no mesmo conjunto reduzido de imagens, essa comparação deve ser interpretada dentro do protocolo experimental adotado.

4.1.1. Desempenho por classe

Tendo em vista o desbalanceamento presente no dataset, também foi analisado o desempenho dos modelos em cada classe individualmente utilizando a métrica de mAP50, que corresponde à média das APs obtidas a partir de um limiar de interseção sobre união (IoU) de 0,5. Assim, pode-se verificar se a redução da resolução afetou de maneira distinta cada uma das categorias avaliadas, especialmente aquelas com menor número de anotações.

Tabela 3. mAP50 por classe dos modelos treinados e avaliados na mesma resolução.

Resolução	car	motorcycle	bus	other
224×224	0.839	0.629	0.746	0.517
192×192	0.798	0.536	0.747	0.467
160×160	0.741	0.439	0.444	0.246
128×128	0.642	0.345	0.375	0.197

A partir dos resultados da Tabela 3, identifica-se que a classe `car` apresentou os maiores valores de mAP50 em todas as resoluções avaliadas, alcançando 0,839 em 224×224 e mantendo desempenho superior às demais classes mesmo em resoluções menores. Isso está associado à maior representatividade da classe no conjunto de dados, bem

como à maior frequência visual de automóveis nas cenas de tráfego urbano analisadas.

A classe `bus` também apresentou desempenho elevado nas maiores resoluções, mas houve uma queda acentuada nas duas menores resoluções, 160×160 e 128×128. Esse resultado sugere que, embora ônibus possuam dimensões maiores nas imagens, sua detecção ainda depende de detalhes suficientes para diferenciar corretamente essa categoria dos demais veículos.

Ademais, as classes `motorcycle` e `other` apresentaram os menores valores de mAP50 ao longo das resoluções avaliadas. No caso das motocicletas, a redução de desempenho pode estar relacionada ao seu menor tamanho visual nas imagens e à maior perda de detalhes quando a resolução é reduzida. Já a classe `other`, por representar uma categoria genérica composta por diferentes tipos de veículos, tende a apresentar maior variabilidade visual, dificultando o aprendizado de padrões consistentes pelo modelo.

4.2. Tempo de inferência no dispositivo embarcado

Além da avaliação offline das métricas de detecção, foi analisado o tempo de inferência dos modelos quando executados diretamente no microcontrolador ESP32-S3, conforme apresentado na Tabela 4. Essa parte do estudo é pertinente, uma vez que a aplicação proposta não depende apenas da qualidade das detecções, mas também da capacidade de executar o modelo em um ambiente embarcado com recursos limitados de processamento e memória. Como limitações, ressalta-se que não foram reportadas medidas de dispersão temporal, como desvio padrão, nem consumo energético.

Tabela 4. Tempo de inferência dos modelos executados no ESP32-S3, com 10 execuções por resolução.

Resolução de entrada	Tempo médio de inferência (s)
224×224	27,381
192×192	21,541
160×160	13,126
128×128	11,721

Com base nos dados obtidos, observa-se que a redução da resolução de entrada diminuiu o tempo necessário para a execução da inferência. Esse comportamento é esperado, uma vez que entradas menores reduzem a quantidade de dados processados pela rede e o custo computacional associado às operações convolucionais.

Apesar disso, os resultados offline indicaram que resoluções menores também reduzem a precisão, a revocação e o mAP50 dos modelos. Logo, embora a diminuição da resolução favoreça a velocidade de execução no dispositivo embarcado, ela impõe uma degradação na qualidade das detecções, especialmente em classes mais sensíveis à perda de detalhes visuais ou com maior variabilidade, como `motorcycle`, `bus` e `other`.

4.3. Análise da relação entre qualidade de detecção e tempo de inferência

A Figura 4 apresenta a variação das métricas de precisão, revocação e mAP50 em função da resolução de entrada dos modelos treinados e avaliados na mesma resolução. Percebe-se que o aumento da resolução favorece o desempenho geral da rede, com o melhor resultado sendo obtido em 224×224 pixels, evidenciando que a preservação de mais detalhes visuais contribui para a classificação e localização dos veículos nas imagens.

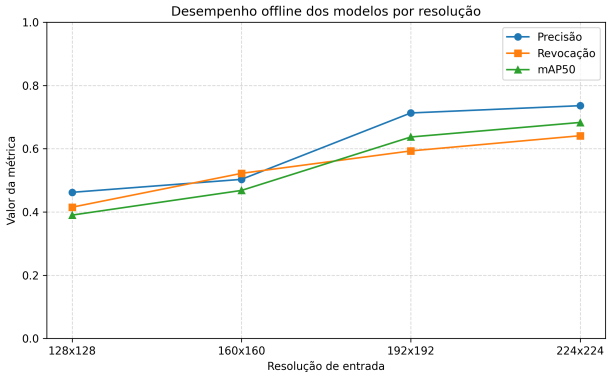


Figura 4. Variação das métricas offline em função da resolução de entrada.

Para analisar conjuntamente qualidade e custo computacional, calculou-se uma eficiência relativa:

$$E = \frac{\text{mAP50}}{T_{\text{inf}}} \tag{1}$$

em que T_{inf} é o tempo médio de inferência. Essa métrica ad hoc foi utilizada apenas como indicador relativo entre as resoluções avaliadas, seus valores absolutos não devem ser interpretados isoladamente. Os resultados evidenciam um compromisso entre qualidade de detecção e custo computacional, com resoluções maiores proporcionando melhor desempenho preditivo, enquanto resoluções menores reduzem o tempo de inferência. Nesse contexto, a configuração de 160×160 pixels representa uma alternativa intermediária entre esses dois fatores, embora a escolha da resolução mais adequada dependa dos requisitos específicos da aplicação.

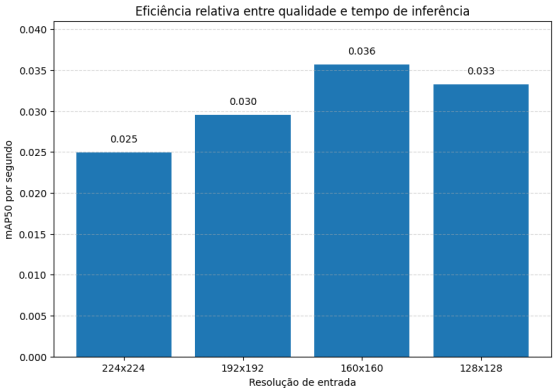


Figura 5. Eficiência relativa entre mAP50 e tempo médio de inferência.

Embora a métrica de mAP50 da resolução de 160×160 seja inferior ao das resoluções de 224×224 e 192×192 pixels, a redução significativa no tempo de execução compensa essa perda. Ressalta-se que essa métrica complementar não substitui a análise individual das métricas de detecção e do tempo de inferência, mas auxilia na identificação da configuração mais adequada para o cenário embarcado deste trabalho.

5. Considerações Finais

O trabalho apresentou a implementação e avaliação de modelos YOLOv5n para detecção de veículos em um dispositivo embarcado ESP32-S3. A partir dos experimentos realizados, verificou-se que a resolução de entrada exerce influência direta tanto na qualidade das detecções quanto no tempo de inferência. Nos testes offline, os modelos treinados e avaliados na mesma resolução apresentaram desempenho superior ao modelo treinado em 640×640 pixels e avaliado em resoluções reduzidas, indicando a importância de adequar o treinamento à resolução efetiva de operação. Entretanto, considerando que a base de dados utilizada possui cerca de 1000 imagens e distribuição desbalanceada entre as classes, a comparação com o modelo treinado em 640×640 deve ser interpretada dentro desse conjunto experimental específico, pois os resultados podem refletir tanto o efeito da resolução quanto limitações associadas ao volume e à distribuição dos dados disponíveis.

Entre as configurações avaliadas, o modelo em 224×224 pixels obteve o melhor desempenho absoluto em termos de mAP50, enquanto a resolução de 160×160 pixels apresentou o menor tempo de inferência entre os modelos com mAP50 superior a 0,45, constituindo, dentro do conjunto experimental, uma alternativa intermediária entre qualidade de detecção e custo computacional. Contudo, como os treinamentos foram realizados em execução única por configuração, ressalta-se que investigações posteriores devem incluir repetições com diferentes sementes e estimativas de variância. Ademais, esse resultado não deve ser generalizado para implementações quantizadas, pois a quantização INT8 pode alterar significativamente a relação entre tempo, memória e acurácia.

Dessa maneira, os resultados demonstram a possibilidade de executar o modelo no ESP32-S3 utilizando o fluxo de conversão adotado, mas evidenciam limitações significativas de desempenho na implementação em ponto flutuante avaliada. No entanto, os tempos de inferência observados entre 11,721 s e 27,381 s por imagem indicam que a implementação avaliada não é adequada para aplicações em tempo real, sendo mais compatível com cenários de amostragem esparsa ou monitoramento de baixa frequência.

Para trabalhos futuros, pretende-se ampliar a avaliação do sistema em cenários reais de operação, utilizando a câmera acoplada ao ESP32-S3 para aquisição de imagens nas vias urbanas em foco. Além disso, podem ser investigadas estratégias de otimização adicionais, como quantização dos pesos, poda da rede e uso de arquiteturas ainda mais leves para dispositivos embarcados. Outra possibilidade consiste em expandir e balancear a base de dados utilizada, de modo a melhorar a capacidade de generalização do modelo. Finalmente, a integração com mecanismos de comunicação sem fio (Wi-Fi ou MQTT) pode permitir o envio das detecções para sistemas externos de monitoramento, expandindo o escopo de uso da solução proposta em sistemas inteligentes de tráfego urbano.

Transparência no Uso de Inteligência Artificial e Considerações Éticas

Durante o desenvolvimento deste trabalho, ferramentas de inteligência artificial generativa foram utilizadas como apoio à revisão textual e aprimoramento da clareza da escrita, bem como de forma auxiliar na elaboração de elementos visuais (Figuras 1 e 3). Todas as contribuições geradas por essas ferramentas foram revisadas e validadas pelos autores, garantindo a coerência técnica e a precisão das informações apresentadas.

Agradecimentos

Trabalho apoiado financeiramente pelo Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) por meio do Programa Institucional de Bolsas de Iniciação Científica (PIBIC) vinculado à Universidade do Estado do Amazonas (UEA).

Referências

Ashkanani, M., AlAjmi, A., Alhayyan, A., Esmael, Z., AlBedaiwi, M., and Nadeem, M. (2025). A self-adaptive traffic signal system integrating real-time vehicle detection and license plate recognition for enhanced traffic management. *Inventions*, 10(1):14.

Brink, H., Richards, J. W., and Fetherolf, M. (2016). *Real-World Machine Learning*. Manning, Shelter Island, NY, 1st edition.

Chang, Y.-H., Wu, F.-C., and Lin, H.-W. (2025). Design and implementation of ESP32-based edge computing for object detection. *Sensors*, 25(6):1656.

Everingham, M., Gool, L. V., Williams, C. K. I., Winn, J., and Zisserman, A. (2010). The Pascal visual object classes (VOC) challenge. *International Journal of Computer Vision*, 88(2):303–338.

Leite, C. and Awad, J. C. M. (2012). *Cidades Sustentáveis, Cidades Inteligentes: Desenvolvimento Sustentável num Planeta Urbano*. Bookman, Porto Alegre, 1st edition.

Loureiro, C. E. N., Guedes, E. B., and Figueiredo, C. M. S. (2025). Monitoramento inteligente de tempo real para tráfego urbano em cidades brasileiras. In *Anais do XVII Simpósio Brasileiro de Computação Ubíqua e Pervasiva (SBCUP)*, pages 41–50, Maceió/AL. SBC.

Moosmann, J., Müller, H., Zimmerman, N., Rutishauser, G., Benini, L., and Magno, M. (2024). Flexible and fully quantized lightweight TinyissimoYOLO for ultra-low-power edge systems. *IEEE Access*, 12:75093–75107.

Prince, S. J. D. (2012). *Computer Vision: Models, Learning, and Inference*. Cambridge University Press, Cambridge.

Sapkota, R. and Karkee, M. (2025). Ultralytics YOLO evolution: An overview of YOLO26, YOLO11, YOLOv8 and YOLOv5 object detectors for computer vision and pattern recognition. arXiv preprint arXiv:2510.09653. Disponível em: `https://arxiv.org/abs/2510.09653`.

Shafi, A. (2025). Running YOLOv5n on an ESP32-S3. Hackster.io. Disponível em: `https://www.hackster.io/asadshafi5/running-yolov5n-on-an-esp32-s3-dd1afc`. Acesso em: 29 maio 2026.

Uber Newsroom (2020). Estudo com dados da Uber conclui que congestionamento piora até 32% quando ocorrem acidentes. Uber Newsroom Brasil. Estudo realizado pelo Instituto Cordial. Disponível em: `https://www.uber.com/pt-BR/newsroom/`. Acesso em: 26 mar. 2026.

Zhao, L., Song, Y., Zhang, C., Liu, Y., Wang, P., Lin, T., Deng, M., and Li, H. (2020). T-GCN: A temporal graph convolutional network for traffic prediction. *IEEE Transactions on Intelligent Transportation Systems*, 21(9):3848–3858.